Connectivity Measures for Internet Topologies[1]

Thomas Erlebach[2]
Department of Computer Science, University of Leicester, UK, t.erlebach@mcs.le.ac.uk
Linda S. Moonen, Frits C.R. Spieksma
Department of Applied Economics, Katholieke Universiteit, Belgium,
{linda.moonen@econ.kuleuven.be, frits.spieksma@econ.kuleuven.be}
Danica Vukadinović[3]
Computer Engineering and Networks Laboratory (TIK), ETH Zürich, Switzerland,
vukadin@tik.ee.ethz.ch

The topology of the Internet has initially been modelled as an undirected graph, where vertices correspond to so-called Autonomous Systems (ASs), and edges correspond to physical links between pairs of ASs. However, in order to capture the impact of routing policies, it has recently become apparent that one needs to classify the edges according to the existing economic relationships (customer-provider, peer-to-peer or siblings) between the ASs. This leads to a directed graph model in which traffic can be sent only along so-called valley-free paths. Four different algorithms have been proposed in the literature for inferring AS relationships using publicly available data from routing tables. We investigate the differences in the graph models produced by these algorithms, focussing on connectivity measures. To this aim, we compute the maximum number of vertex-disjoint valley-free paths between ASs as well as the size of a minimum cut separating a pair of ASs. Although these problems are solvable in polynomial time for ordinary graphs, they are $\mathcal{NP}$-hard in our setting. We formulate the two problems as integer programs, and we propose a number of exact algorithms for solving them. For the problem of finding the maximum number of vertex-disjoint paths, we discuss two algorithms; the first one is a branch-and-price algorithm based on the IP formulation, and the second algorithm is a non LP based branch-and-bound algorithm. For the problem of finding minimum cuts we use a branch-and-cut algorithm, based on the IP formulation of this problem. Using these algorithms, we obtain exact solutions for both problems in reasonable time. It turns out that there is a large gap in terms of the connectivity measures between the undirected and directed models. This finding supports our conclusion that economic relationships need to be taken into account when building a topology of the Internet.

[1]This research was partially supported by the European Commission Thematic Network APPOL II (IST-2001-32007), with funding for the Swiss partners provided by the Swiss Federal Office for Education and Science (BBW).

[2]Work partly done while this author was with the Computer Engineering and Networks Laboratory at ETH Zürich.

[3]Supported in DICS-Project No. 1838, *Robustness of the Internet at the Topology and Routing Level*, by the Hasler Foundation.

1. Introduction

It is a cliché to state that stability and robustness of the Internet are fundamental for securing today's efficient communication. Maintaining the speed and the reliability of Internet-based communication is a prime challenge for service providers, their clients, and other involved institutions. In order to understand the potential vulnerability of Internet-based communication, we need to get an idea of the routes that are being used for sending traffic, of the routes that could be used for sending traffic, and how different ways of sending traffic vary with respect to their susceptibility to failing servers and/or failing connections.

A first step is then retrieving how traffic is being sent over the Internet. This, however, is already not so easy to find out (Chang et al., 2004). To explain this, let us view the Internet as a set of Autonomous Systems (ASs; an AS is a subnetwork under separate administrative control), which are connected by physical links. ASs exchange routing information using the Border Gateway Protocol (BGP); this is a protocol that governs the communication between a pair of ASs. More specifically, each AS uses a local routing policy that determines which routes are announced to which neighboring ASs. For commercial reasons, details about these local policies of individual ASs are not publicly available. Obviously, this makes it difficult to create an accurate model that can be used in the analysis of the robustness of the Internet.

The goal of this chapter is to contribute to the development of an accurate model for the Internet topology. We do this by comparing different methods that have been proposed in the literature to infer the topology of the Internet using observed traffic-data. The comparison focusses on two connectivity measures, namely the number of (disjoint) paths between a given pair of ASs, and the size of a minimum cut separating a pair of ASs. Let us proceed by describing the relevant issues in more detail.

Routing policies depend mostly on the economic relationships between ASs. They represent an important aspect of Internet structure. Huston (1999a, 1999b) discussed the main trends in the diversity of commercial agreements between ASs. We will refer to local policies governed by the BGP as BGP routing policies, or BGP policies for short. The impact of economic relationships on the engineering level, more precisely on BGP policies, has been recognized as one of the reasons for BGP path inflation (i.e., the phenomenon that traffic uses paths that are much longer than necessary; see Gao and Wang (2002)) and one of the important factors in route convergence analysis (i.e., the fact that, when a previously valid path to a destination D becomes invalid, it takes a long time until the network has obtained a new valid path to D (Labovitz et al., 2001)). Thus, the previously adopted undirected model of the Internet, which ignores BGP policies, is only a crude approximation of reality and might produce a distorted picture of the

routes used in practice. On the other hand, incorporating all of the peculiarities of the manifold contracts between ASs in a new model would add too much complexity (assuming one would know these contracts). Therefore, a coarse classification of AS relationships into three categories—customer-provider, peer-to-peer and siblings—has been proposed (Gao, 2001). More recent work has restricted attention to customer-provider and peer-to-peer relationships only (Subramanian et al., 2002).

If ASs A and B are in a customer-provider relationship, i.e., if A is a customer of B, then B announces all its routes to A, but A announces to B only its own routes and routes of its own customers. If they are peers, they exchange their own routes and routes of their customers, but not routes of their providers or other peers. If ASs A and B are siblings, then A announces all its routes to B and B announces all its routes to A. These policies arise because customers do not want to act as transit ASs for their providers, i.e., a provider cannot route traffic through a customer to a different provider of that customer. As a consequence, only *valley-free* paths are valid, i.e., paths that first go "up" in the hierarchy and then "down" towards the destination. (A formal definition will be given in Section 2. In this chapter we will use the terms "valley-free path" and "valid path" interchangeably.)

Thus, one arrives at the following model. The Internet is a graph containing the ASs as vertices. The graph can have directed and undirected edges. There are three different ways in which two vertices A and B can be connected:

i) an undirected edge between A and B. This is interpreted as "A and B are peers."

ii) a directed edge from A to B, and a directed edge from B to A. This is interpreted as "A and B are siblings."

iii) a directed edge from A to B, and no directed edge from B to A. This is interpreted as "A is customer of B."

The resulting graph is called a ToR graph (see Section 2 for formal definitions). Two ASs with at least one physical link between them are connected by a single edge (or a single pair of edges, in the case of siblings) in this model, no matter how many physical links there are between these two ASs. For comparison, note that the previously adopted *undirected graph model* of the Internet consisted of an undirected graph with an undirected edge between two ASs if there is at least one physical link between them.

Since information about the economic relationships between ASs is not publicly available (such information is often treated like a business secret), four algorithms have been proposed for inferring these relationships from BGP routing table information (Gao, 2001; Subramanian et al., 2002; Di Battista et al., 2003; Erlebach et al., 2002). However, it is not known

how good the topologies produced by these algorithms are and how these topologies differ from each other. Therefore, in view of the large impact of BGP policies in the Internet, we perform a thorough comparison of these graph models in this chapter. Since the main effect of BGP policies is that they restrict the set of paths that traffic can take in the network, we consider mainly path-related criteria. In particular, we compute the maximum number of vertex-disjoint valley-free paths between two ASs and the minimum number of vertices that must be removed from the graph so that no valley-free path between these two ASs remains. These are natural adaptations of classical measures of connectivity in graphs to the valley-free path model. It is well known that in the standard graph models (in the standard model, a path consists of a sequence of forward arcs in the directed case and of a sequence of undirected edges in the undirected case) the maximum number of disjoint paths between s and t is equal to the size of a minimum s-t cut (provided that s and t are not adjacent); moreover, the corresponding solutions can be computed efficiently (see Ahuja et al. (1993)). In a ToR graph this is not the case. It is $\mathcal{NP}$-hard to compute the maximum number of vertex-disjoint s-t paths; it is also $\mathcal{NP}$-hard to compute the minimum size of a valid s-t cut. The best known approximation ratio is 2 for both problems. Also, the minimum size of an s-t cut can be up to twice the maximum number of vertex-disjoint s-t paths. Thus, the max-flow min-cut equality holds only approximately for ToR graphs (Erlebach et al., 2005). We are able to obtain optimal solutions with a moderate amount of computation time using exact approaches, one of which involves applying a branch-and-price algorithm. We compare the results from the exact methods with those of the 2-approximation algorithms from Erlebach et al. (2005).

We also compute disjoint paths and minimum cuts in the undirected Internet graph and compare the results with those of the different directed models. Furthermore, we investigate directed customer-provider cycles, which are a somewhat unexpected structure, in the directed models. We claim that these cycles can help to detect misclassified relationships and thus improve the accuracy of the Internet topology. We also give statistics about the minimum and maximum length of the observed cycles. Finally, we report statistics concerning the number of ASs that are connected by directed paths in the different directed models, a quantity that is related to the depth of the provider hierarchy and to the *customer-preference* aspect of current inter-domain routing (Feigenbaum et al., 2002). The latter means that paths through customers are preferred over paths through peers, and these to paths through providers.

In summary, our investigations address the following research questions:

- Do the differences between the undirected graph model and the directed graph models with respect to connectivity properties confirm the importance of incorporating BGP policies in the model?

- How do the directed graph models produced by the four algorithms proposed by Gao (2001), Subramanian et al. (2002), Di Battista et al. (2003), and Erlebach et al. (2002) compare to each other? Here, we are mainly interested in comparing connectivity measures and the depth of the provider hierarchy.

- How many directed customer-provider cycles occur in the different directed graph models, and can they be helpful in the detection of misclassified edges?

- In a graph on the scale of the Internet (containing up to 11,000 vertices and 30,000 edges), is it feasible to compute exact solutions to the $\mathcal{NP}$-hard problems of finding a maximum number of vertex-disjoint valley-free paths between two ASs or the size of a minimum cut? Such computations could prove useful in future investigations of robustness issues of the Internet.

- How does the performance of the 2-approximation algorithms proposed by Erlebach et al. (2005) compare to the performance of the exact algorithms, both in the quality of the solutions found and in the running times?

1.1 Related work and motivation

Our starting point for the interpretation of BGP policies is the work of Gao (2001) that addressed the problem of unavailable information about the exact relationships between ASs. A heuristic algorithm was proposed for inferring AS relationships from BGP routing tables. In addition, it was observed that a path between a pair of ASs follows a particular structure: no path contains more than one peer-to-peer relationship, and once a provider-customer or peer-to-peer relationship is encountered in the path, no customer-provider relationship can follow. If we ignore sibling relationships for the moment and imagine that providers are at a higher level than their customers and peers are at the same level, the valid paths are "only up," "only down," or "first up and then down." Valid paths can have only one "peak" (which can consist of a single AS or of two ASs connected by a peer-to-peer relationship) and they must not contain "valleys." Therefore, such paths are also called *valley-free* paths. We use the same characterization of valid paths in this chapter.

Further work trying to infer AS relationships is presented by Subramanian et al. (2002). They formalize the problem by posing it as the optimization problem of giving an orientation to the edges of an undirected AS graph with the objective of maximizing the number of paths in the given BGP tables that become valid for this orientation. They pose the complexity of this problem as an open question. They also give a heuristic algorithm that

infers relationships by first ranking all ASs and then applying certain rules to decide about the relationships between pairs of ASs using the rank values. Independently obtained results from Di Battista et al. (2003) and Erlebach et al. (2002) resolve the open question of Subramanian et al. (2002) and prove this inference problem to be $\mathcal{NP}$-hard. Two heuristic algorithms for calculating approximately optimal orientations with respect to the number of valid paths are also presented by Di Battista et al. (2003) and Erlebach et al. (2002), respectively.

Rimondini et al. (2004) compare the algorithms from Subramanian et al. (2002) and Di Battista et al. (2003) with respect to two measures. First, the AS relationships that are found by a certain algorithm on data sets from different moments in time are considered (called *stability* analysis in the paper). Second, the AS relationships found by the two algorithms on the same data set are taken into account (referred to as *algorithm independence* analysis). They conclude that both algorithms produce highly stable results, and that the AS relationships found by both algorithms are very similar. This leads the authors to the conclusion that the valley-free path approach leads to reliable results.

In another paper by Xia and Gao (2004), a comparison of the algorithms from Gao (2001) and Subramanian et al. (2002) is done. Also, in this chapter, a new algorithm for inferring AS relationships is proposed, which is also taken into account in the comparison. The authors evaluate the accuracy of the three algorithms using partial AS relationships obtained from BGP community attribute data and IRR (Internet Routing Registry) databases. They conclude that the new algorithm proposed in the paper outperforms the algorithms from Gao (2001) and Subramanian et al. (2002).

In this chapter, we compare the AS relationships computed by all four algorithms proposed by Gao (2001), Subramanian et al. (2002), Di Battista et al. (2003), and Erlebach et al. (2002), and we try to identify important characteristics of the relationship classifications produced by these algorithms. The motivation for our investigations comes from several papers that showed the impact of BGP policies on important features of Internet routing such as path inflation and routing convergence (see Labovitz et al. (2001), and Tangmunarunkit et al. (2001a, 2001b, 2003). In addition, recent results about measurements on the AS level of the Internet have shown that there is a need for a simple and accurate algorithm to infer relationships; see Spring et al. (2003) about path inflation in inter- and intra-domain routing, Akella et al. (2003a) about multi-homing (i.e., the phenomenon that customers tend to have more than one external link to different providers, in order to guarantee the reliability of their network), and Akella et al. (2003b) about scaling properties of the Internet regarding link congestion.

Teixeira et al. (2003) compute the number of vertex- and edge-disjoint paths for the undirected model of the Internet AS topology, as well as for the topology of one Internet Service Provider. They did not take routing policies

into account. Here, we investigate how the number of vertex-disjoint paths and the size of a minimum cut can be computed exactly and in reasonable time for Internet graphs that are constrained by BGP policies. These results may be helpful for future research on more resilient and efficient inter-domain routing.

1.2 Outline

Section 2 gives formal definitions of the concepts that we require in this chapter, and we discuss a primal-dual formulation of the problem. Section 3 deals with the computation of vertex-disjoint valid paths; Section 4 describes how we compute minimum cuts with respect to valid paths. Section 5 reviews the known 2-approximation algorithms for solving both problems. In Section 6, we present our experimental results concerning the number of vertex-disjoint valid paths and the sizes of minimum cuts in the four different models with inferred relationships and the undirected model. We discuss their implications and also the differences that we observe in the depth of the provider hierarchy in the different models. Statistics about directed cycles in the graphs are given and some examples where they can be used to detect misclassifications are shown. We conclude in Section 7 by summarizing our results and drawing conclusions.

2. Problem description

In order to formulate the problem, we first state some preliminaries in Section 2.1. Then, in Section 2.2 we give a mathematical formulation for the problems of finding the maximum number of vertex-disjoint paths and minimum cut sizes.

2.1 Preliminaries

Subramanian et al. (2002), Di Battista et al. (2003), and Erlebach et al. (2002) refer to the problem of inferring the AS relationships in the Internet as the *Type of Relationship (ToR) problem*. Following this terminology, we construct a graph $G = (V, E)$, called a *ToR graph*, as follows: the vertices of G are the ASs. As mentioned before, a directed edge from u to v, where $u, v \in V$, together with a directed edge from v to u means that u and v are siblings. A directed edge from u to v means that u is a customer of v, and an undirected edge means that u and v are in a peer-to-peer relationship. In a ToR graph, a directed edge from u to v is denoted by (u, v), and an undirected edge between u and v by $\{u, v\}$.

We define a path $p = (v_1, v_2, \ldots, v_r)$ from v_1 to v_r in a ToR graph $G = (V, E)$ to be *valid* if it satisfies one of the two following conditions:

1. There exists some j, $1 \leq j \leq r$, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq j-1$ and $(v_i, v_{i-1}) \in E$ for $j+1 \leq i \leq r$.

2. There exists some j, $1 \leq j \leq r$, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq j-1$, $\{v_j, v_{j+1}\} \in E$, and $(v_i, v_{i-1}) \in E$ for $j+2 \leq i \leq r$.

Otherwise, a path is called *invalid*. This definition of valid paths captures the notion of "valley-free" paths arising from BGP routing policies. From now on, whenever we talk about paths in a ToR graph, we refer to valid paths. A path from s to t is also called an *s-t path*. Note that the reverse of an s-t path is a t-s path, hence the direction of a valid path is not important. Two s-t paths are called *vertex-disjoint* if they do not share any vertices except s and t.

Let $p = (v_1, v_2, ..., v_r)$ be a valid path from s to t. We can divide p into a forward part and a backward part at some node v_j, such that $(v_i, v_{i+1}) \in E$, $i = 1, 2, \ldots, j-1$ (we know by definition that such a j exists; if j is not unique, we simply choose the maximal value for j). If p contains only directed edges, we say that a node v_l is on the forward part of p if $l < j$, v_l is on the backward part of p if $l > j$ and v_l is the node where p changes direction if $l = j$. If p contains an undirected edge, we say that a node v_l is on the forward part of p if $l \leq j$ and v_l is on the backward part if $l > j$.

Let $G = (V, E)$ be a ToR graph. For two non-adjacent vertices s and t in G, a *minimum valid s-t cut* in G is a set of vertices $C \subseteq V \setminus \{s, t\}$ of minimum cardinality such that there is no s-t path in the ToR graph $G \setminus C$ (i.e., in the graph that is obtained from G by deleting the vertices in C and their incident edges). Note that a minimum valid s-t cut is a smallest set of ASs whose failure disconnects s and t if only valid paths are allowed.

A *directed cycle* $v = (v_1, v_2, \ldots, v_r)$, $r > 2$, in a ToR graph $G = (V, E)$ is defined in the usual sense, i.e., the vertices $v_1, v_2, \ldots, v_r$ are distinct and we have $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, r-1$ and $(v_r, v_1) \in E$.

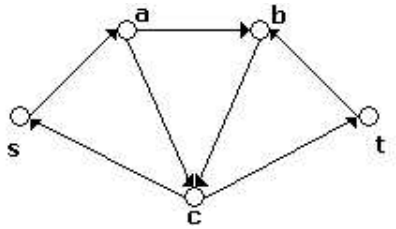

Figure 1: Gap between number of disjoint paths and minimum cut size.

As mentioned before, the maximum number of vertex-disjoint paths can be strictly less than the number of nodes in a minimum cut; we give an example from Erlebach et al. (2005) to illustrate this. In Figure 1 we see that the maximum number of vertex-disjoint paths is equal to 1, while the size of

a minimum cut equals 2. Indeed, one can verify that the set of valid s-t paths equals $\{(s,a,b,t),(s,a,b,c,t),(s,a,c,t),(s,a,c,b,t),(s,c,b,t)\}$, and thus the maximum number of vertex-disjoint paths is equal to 1. Furthermore, one can easily verify that a minimum cut has at least size 2, since after removing one of the nodes a, b or c, there is still a valid path connecting s and t.

2.2 Problem formulation

Let us now give two integer programming formulations; the first formulation (denoted by P) models the problem of finding a maximum number of vertex-disjoint paths between s and t, the second formulation (denoted by D) models the problem of finding a minimum-sized set of nodes such that each path between s and t contains at least one node from this set.

Let $G = (V, E)$ be a ToR graph and s, t two distinct vertices of G. Assume that there is no direct edge between s and t (otherwise, we remove the direct edge, compute the maximum number of vertex-disjoint s-t paths, and add one to the result). Denote by $\mathcal{P}$ the set of all valid s-t paths in G, and let $\mathcal{V}_p$ be the set of all vertices contained in path $p \in \mathcal{P}$, except for s and t. Further, we define a decision variable x_p for each valid path p, as follows:

$$x_p = \begin{cases} 1 & \text{if valid path } p \text{ is in the solution} \\ 0 & \text{otherwise.} \end{cases}$$

Using a set-packing formulation, we get the following integer programming formulation:

$$\text{(P)} \quad \max \sum_{p \in \mathcal{P}} x_p \tag{1}$$

$$s.t. \sum_{p: v \in \mathcal{V}_p} x_p \leq 1 \qquad \forall v \in V \setminus \{s,t\} \tag{2}$$

$$x_p \in \{0,1\} \qquad \forall p \in \mathcal{P} \tag{3}$$

The objective (1) is to maximize the number of paths between s and t. Constraints (2) state that each vertex (except for s and t) can belong to at most one path, and constraints (3) are the zero-one constraints on the x_p variables.

The second formulation has a variable y_v for every $v \in V \setminus \{s,t\}$:

$$y_v = \begin{cases} 1 & \text{if vertex } v \text{ is in the } s\text{-}t \text{ cut} \\ 0 & \text{otherwise.} \end{cases}$$

The second formulation can now be given as follows:

$$\text{(D)} \quad \min \sum_{v \in V \setminus \{s,t\}} y_v \tag{4}$$

$$s.t. \sum_{v \in \mathcal{V}_p} y_v \geq 1 \qquad \forall p \in \mathcal{P} \tag{5}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \setminus \{s,t\} \tag{6}$$

A property of formulations (P) and (D) is that the LP-relaxation of (P) and the LP-relaxation of (D) constitute a primal-dual pair of linear programs. Further, notice that formulation (P) has exponentially many variables (since the number of valid s-t paths can be exponential in the number of vertices), and, equivalently, formulation (D) has exponentially many constraints.

3. Problem P: vertex-disjoint paths in ToR graphs

In this section we present two exact algorithms for solving problem P; i.e., for finding the maximum number of vertex-disjoint paths in ToR graphs. The first one is a branch-and-price algorithm based on the integer programming formulation (1)-(3) (Section 3.1), and the second algorithm is a branch-and-bound method in which a max-flow computation has to be performed in each node of the search tree (Section 3.2).

3.1 A branch-and-price algorithm

Branch-and-price is a technique for solving integer programs with a huge number of variables. We refer to Barnhart et al. (1998) or Vanderbeck and Wolsey (1996) for a thorough description of this technique. Here we apply it to solving instances of formulation (P). There are (at least) two important issues to be considered when developing a branch-and-price algorithm: (i) how to solve the pricing problem (this enables us to conclude that either we have solved the LP-relaxation of (P), or we have identified a new variable (column) to be added to the restricted master; (ii) how to branch. We need to develop a partition of the solution space in such a way that the efficient solvability of the pricing problem is preserved.

3.1.1 Column generation

We start by generating a feasible solution (consisting of a set of vertex-disjoint paths) as follows. We apply a simple breadth-first search to find a valid path between s and t, we add this path to the solution, and remove all

nodes in this path (except s and t) from our graph. Then a new iteration starts, and we repeat the breadth-first search until no more valid paths can be found. The resulting set of paths found by this *iterated breadth-first search* is denoted by $\mathcal{P}'$, and its value (number of disjoint paths) is referred to as $V_{\mathcal{P}'}$. We consider the restriction of the LP-relaxation of (P) to the variables x_p for $p \in \mathcal{P}'$ (the *restricted master problem*). We solve the restricted master using an LP-solver and obtain a solution to the restricted primal program and its corresponding dual. Let us call the dual solution y^*. Now, we need to check whether y^* is also a feasible solution to the dual program that includes constraints for all paths $p \in \mathcal{P}$. In other words, we need to check whether there exists a valid s-t path p in the graph such that $\sum_{v \in p} y^*_v < 1$. This problem is known as the *pricing problem* (Vanderbeck and Wolsey, 1996). We can solve the pricing problem in polynomial time, thereby implying that the LP-relaxation of formulation (P) (as well as the LP-relaxation of (D)) can be solved in polynomial time.

Claim 1. *The LP-relaxation of* (P) *can be solved in polynomial time.*

Proof. We prove the claim by showing that the pricing problem can be reduced to a shortest path problem. The result then follows from the "separation = optimization" result (Grötschel et al., 1988).

Consider the so-called 2-layer graph that has been proposed by Erlebach et al. (2005) (we first assume that there are no undirected edges in G): two copies G_1 and G_2 of graph G are created, but in G_2 all edge-directions are reversed. Then, so called "vertical edges" are added, i.e., directed edges from the vertices in G_1 to their copies in G_2. Finally, s in G_1 is identified with its copy in G_2 and all edges that end in s are removed, and t in G_1 is identified with its copy in G_2 and all edges that start in t are removed. In this way, all valid s-t paths in G correspond to directed paths from s to t in the 2-layer model: If a valid s-t path in G first uses some forward edges and then some backward edges, its forward part in the 2-layer model lies in G_1, then it switches to the second layer using a vertical edge, and then it goes again forward to t in G_2 because of the inverted edge-directions. (See Figure 2 for an illustration.)

We can deal with undirected edges in the following way. For an undirected edge $\{a, b\}$, we do not add corresponding edges to G_1 or G_2, but instead add directed edges (a_1, b_2) and (b_1, a_2) to the 2-layer model, where a_1, a_2 (b_1, b_2) are the copies of a (b) in G_1 and G_2, respectively. This ensures that valid s-t paths in G that include an undirected edge also have a corresponding path in the 2-layer model.

Next, we define the edge weights of the 2-layer graph as follows: edges entering a copy of vertex v get weight y^*_v, except for vertical edges, which get weight 0. Observe that a shortest directed path from s to t in the 2-layer model gives us a valid s-t path in G that minimizes the sum of the y^*_v values

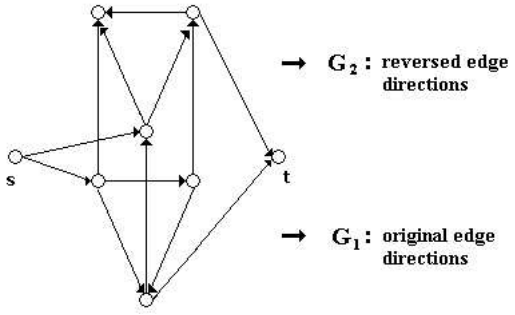

Figure 2: The 2-layer graph of the ToR graph depicted in Figure 1.

of the vertices on the path. Since the shortest path problem can be solved in polynomial time (using for instance Dijkstra's algorithm), we can solve the pricing problem in polynomial time. □

If the solution of the pricing problem produces a valid s-t path p such that the sum of y_v^* values on this path is less than 1, we add path p to the restricted master and repeat the procedure. If there is no such path, we are done and have obtained an optimal solution to the LP-relaxation of (P). If the obtained solution is fractional, i.e., contains variables whose values are strictly between 0 and 1, we use a branching strategy in order to arrive at an integral solution.

3.1.2 Branching

If the optimal solution to the linear programming relaxation is fractional, a natural approach is to try different ways of fixing these variables to integers and solving the problem recursively for each of these alternatives (branching). Here it is important to preserve the form of the pricing problem and its efficient solvability in the branching procedure. We achieve this as follows.

Given a feasible, optimal solution x^* to the LP-relaxation of (P), we call a vertex *fractional* if it has at least three incident edges that lie on different valid paths with value $x_p^* > 0$. Notice that if a solution is fractional, it has at least one fractional vertex. Our branching strategy is as follows: for a fractional vertex w, we delete all edges incident to w except two that could lie consecutively on some valid path. Each possible way of doing this forms a branch. Thus, for instance, if w has k incoming edges and ℓ outgoing edges, the number of branches is $k\ell + \binom{k}{2}$. If there are many fractional vertices, we choose one for branching that has a maximum number of incident edges lying on fractional paths.

In this way we exclude the current fractional solution, but do not exclude any integral solution, and the problem structure is preserved: in each branch,

we solve a problem of the same type on a graph with fewer edges.

For each branch, if the value of the fractional solution is not larger than the value of the best integral solution found so far, we do not enter that branch. Otherwise, we explore all branches in a depth-first traversal. In this way we are sure to arrive at an optimal integral solution to (P).

We remark that our approach can be adapted easily to a version of the problem where each vertex v has an integral capacity c_v and we allow up to c_v valid paths passing through it. (Here, valid paths could occur more than once in the solution.) To solve this version of the problem, we simply replace each vertex v by c_v copies and then apply our algorithm as described above.

The branch-and-price algorithm for a given a ToR graph G and two distinct vertices s and t is summarized by the pseudo-code given in Figure 3.

3.1.3 Valid inequalities

In order to strengthen the LP-relaxation, a natural strategy is to add valid inequalities. In this section we will discuss a class of inequalities that is valid for formulation (P) of the vertex-disjoint paths problem. We will refer to these inequalities as *triangle inequalities*.

As the name suggests, we consider triangles in the ToR graphs. We define a triangle as a subset of three vertices which are connected with customer-provider edges in such a way that they do not form a directed cycle. For example, if there are three vertices a, b and c, and there is an edge from a to b, an edge from a to c and a third edge from b to c, this is a triangle. For each such triangle $t = (a, b, c)$, we define $T_{abc} = \{p \in \mathcal{P} |\ p \text{ contains } \{a, b\} \text{ or } \{a, c\} \text{ or } \{b, c\}\}$.

Now, for every triangle t in a ToR graph the following inequality states that the sum of all valid paths using one of the three edges in t must be smaller than or equal to one:

$$\sum_{p:p\in T_{abc}} x_p \leq 1 \qquad \forall \text{ triangles } (a, b, c) \in V^3 \tag{7}$$

It is clear that inequalities (7) are valid for (P). One can view inequalities (7) (as well as inequalities (2)) as a manifestation of clique-inequalities for the node packing problem. Indeed, when we build a graph in which there is a node for every path $p \in \mathcal{P}$, and where two nodes are connected iff the two corresponding paths share a vertex in the ToR graph, it is obvious that the node packing problem on this graph is exactly problem (P). Notice that inequalities (2) and (7) need not constitute all clique inequalities in the node packing graph. In Section 6.2 we report shortly on the computational effectiveness of these inequalities.

BRANCH-AND-PRICE ALGORITHM VERTEXDISJOINTPATHS

1. Calculate an initial solution consisting of a set of paths $\mathcal{P}'$ with value $V_{\mathcal{P}'}$ using the iterated breadth-first search, and let $V^* = V_{\mathcal{P}'}$. Create a list L and add to L a branching node corresponding to the input graph G.

2. $L = \varnothing$?

   YES: STOP. An optimal solution is found with value V^*.

   NO: Select the next branching node G from L (i.e., the branching node that was added most recently to L), remove it from L, calculate a set $\mathcal{P}'$ of edge-disjoint s-t paths in G using iterated breadth-first search, and continue with step 3.

3. Solve the LP-relaxation using only those variables that correspond to a path in $\mathcal{P}'$.

4. Solve the pricing problem. Is there a variable (a path) with negative reduced costs?

   YES: Add this variable to $\mathcal{P}'$ and go to step 3.

   NO: An optimal solution to the LP-relaxation is found with value V_{LP}. Continue with step 5.

5. $V_{LP} > V^*$?

   YES: Continue with step 6.

   NO: Go to step 2.

6. Is the solution to the LP-relaxation integral?

   YES: $V^* = V_{LP}$. Go to step 2.

   NO: Select a fractional vertex v. For each possible way of deleting all edges incident to v except for two edges that could lie consecutively on some valid path, create a new branching node (i.e., the graph obtained by deleting the respective edges) and add it to L. Go to step 2.

Figure 3: Pseudo-code of the branch-and-price algorithm.

3.2 A branch-and-bound algorithm

Our second algorithm for solving the vertex-disjoint paths problem is a non LP-based branch-and-bound algorithm, in which we use the same 2-layer graph representation as explained in Section 3.1.

We start with an initial solution, computed by the iterated breadth-first search discussed in Section 3.1. The value of this solution is a lower bound on the integer optimum. Next, we compute a maximum flow in the 2-layer graph, where we assign a capacity of 1 to each vertex. The flow we find is not necessarily vertex-disjoint (since it may happen that the maximum flow found uses a node in G_1 and its copy in G_2), so it is an upper bound on the optimal solution. We first check whether the flow found by the maximum flow procedure is vertex-disjoint, or equal to the lower bound, in which case we have found the integer optimum. Otherwise, we have to branch, which we do as follows:

In every node in the search tree, we select a vertex v from the original graph that is used more than once in the flow found by the maximum flow procedure. This vertex v has a copy v_1 in the first layer, and a copy v_2 in the second layer of the 2-layer graph. Now, we generate two new branches as follows:

In the first branch, we delete vertex v_1, and all its adjacent edges, from the 2-layer graph. In the second branch, we delete all incoming edges of v_2, except for the vertical edge (v_1, v_2), from the 2-layer graph. Next, we perform a maximum flow calculation in each branching node, and repeat this procedure until we have found the integer optimum. The correctness of the branching step follows from the observation that if a node occurs in a path of the solution, it is either on the backward part of the path, which is permitted in the first branch, or it is on the forward path or it is the node where the path changes direction (see Section 2.1), which is permitted in the second branch.

The branch-and-bound algorithm is summarized by the pseudo-code given in Figure 4. In our implementation, we actually compute min-cost maximum flows (all edges are assigned a cost of one) instead of standard maximum flows, as we expect that maximum flows using a minimum total number of edges can reduce the number of branching nodes required.

4. Problem D: minimum cuts in ToR graphs

We solve the minimum cut problem using the dual (D) presented in Section 2.2. Since (D) might have exponentially many constraints, we first compute a set $\mathcal{P}'$ of vertex-disjoint s-t paths using the iterated breadth-first search as described in Section 3.1 and start by solving the LP-relaxation of (D) using only the constraints for paths in $\mathcal{P}'$. Solving this small linear

BRANCH-AND-BOUND ALGORITHM VERTEXDISJOINTPATHS

1. Calculate an initial solution consisting of a set of paths $\mathcal{P}'$ with value $V_{\mathcal{P}'}$ using iterated breadth-first search, and let $V^* = V_{\mathcal{P}'}$. Create a list L and let $L = \varnothing$.

2. Construct the 2-layer graph H, and add to L a branching node corresponding to H.

3. $L = \varnothing$?

   YES: STOP. An optimal solution is found with value V^*.

   NO: Select the next node H from L (i.e., the branching node that was added most recently to L), remove this node from L and continue with step 4.

4. Calculate a maximum flow MF with value V_{MF} in the 2-layer graph H.

5. $V_{MF} > V^*$?

   YES: Continue with step 6.

   NO: Go to step 3.

6. Does the maximum flow MF in H correspond to vertex-disjoint paths in G?

   YES: $V^* = V_{MF}$. Go to step 3.

   NO: Select a vertex v that is used more than once in MF. Create two new branching nodes as follows:

   (a) Delete copy v_1 of v from the 2-layer graph.

   (b) Delete all incoming edges of copy v_2 of v from the 2-layer graph, except for the edge (v_1, v_2).

   Add the branching node corresponding to each of these two branches to L. Go to step 3.

Figure 4: Pseudo-code of the branch-and-bound algorithm.

program gives us a solution y^*. Then we check whether there is a valid path such that $\sum_{v \in p} y^*_v < 1$, again using a shortest-path algorithm in the 2-layer model of the graph (i.e., we solve the separation problem with respect to constraints (5)). If such a path is found, we add the corresponding constraint to our linear program (D) and repeat the procedure until no more valid paths with $\sum_{v \in p} y^*_v < 1$ can be found. The resulting solution y is an optimal solution to the LP-relaxation of (D). In case the resulting solution y is fractional, we branch.

The branching is more straightforward than for the vertex-disjoint paths problem. If there is a vertex v such that $0 < y_v < 1$, we add a constraint $y_v = 0$ (an *exclusion* constraint) in one branch and $y_v = 1$ (an *inclusion* constraint) in the other branch to the linear program and solve it again, thus having two branches for a fractional vertex. If there are many fractional vertices, we simply branch on the first one that we find. Similarly to the previous case, we do not enter a branch where the optimal fractional solution is at least as large as the smallest integral solution found so far. The branch-and-cut algorithm is summarized by the pseudo-code given in Figure 5.

We remark that the same approach can be used to solve the generalization of the problem where each vertex v has a weight w_v and the objective is to find a valid s-t cut of minimum weight. The only difference is that the objective function becomes $\min \sum_{v \in V \setminus \{s,t\}} w_v y_v$.

Notice that we use two different approaches for solving the linear programming relaxations of formulations (P) and (D). We found that there were no significant differences in running-time between the two approaches.

5. Approximation algorithms

In this section we discuss two 2-approximation algorithms for finding the maximum number of vertex-disjoint paths and the size of minimum cuts. Both algorithms are presented by Erlebach et al. (2005). In order to make the presentation self-contained, we repeat the description of these algorithms in this section. Section 5.1 deals with the algorithm for the problem of finding the maximum number of vertex-disjoint paths, and in Section 5.2 we give the algorithm for calculating the size of a minimum cut.

5.1 Vertex-disjoint paths

Before stating the approximation algorithm, we need some definitions. If the forward part of a path p_1 intersects a backward part of a path p_2 at a node v, we speak of a crossing at v. The two paths p_1 and p_2 can be recombined at the crossing to form a new path, consisting of the first part of p_1 and the last part of p_2. Given a graph $G = (V, E)$ and two vertices s and t, the algorithm is as follows.

BRANCH-AND-CUT ALGORITHM MINCUT

1. Let $V^* = \infty$. Create a list L and add to L a branching node corresponding to an empty set of inclusion/exclusion constraints.

2. $L = \varnothing$?

   YES: STOP. An optimal solution is found with value V^*.

   NO: Select the next node C from L (i.e., the branching node that was added most recently to L), remove this node from L, calculate a set of vertex-disjoint s-t paths $\mathcal{P}'$ using iterated breadth-first search, and continue with step 3.

3. Solve the LP-relaxation using only the constraints that correspond to a path in $\mathcal{P}'$ and the inclusion/exclusion constraints from C.

4. Solve the separation problem. Is there a variable (a path) with negative reduced costs?

   YES: Add this variable to $\mathcal{P}'$ and go to step 3.

   NO: An optimal solution to the LP-relaxation is found with value V_{LP}. Continue with step 5.

5. $V_{LP} < V^*$?

   YES: Continue with step 6.

   NO: Go to step 2.

6. Is the solution to the LP-relaxation integral?

   YES: $V^* = V_{LP}$. Go to step 2.

   NO: Select a vertex v such that y_v is fractional. Create two new branching nodes as follows:

   - In the first node, add the exclusion constraint $y_v = 0$ to C.
   - In the second node, add the inclusion constraint $y_v = 1$ to C.

   Add these two nodes to L.

Figure 5: Pseudo-code of the branch-and-cut algorithm.

2-Approximation algorithm DisjointPaths (Erlebach et al., 2005)

1. Construct the 2-layer graph, and calculate a maximum flow in this graph.
2. Add, for each path in this maximum flow, the corresponding path in the original graph G to $\mathcal{P}'$.
3. Define $\mathcal{F}$ as the set of all forward parts of paths in $\mathcal{P}'$, and $\mathcal{B}$ as the set of all backward parts.
4. Label all forward parts and all crossings as unscanned. Recombine the forward and backward parts as follows:
   (a) Select an unscanned forward part p_f from $\mathcal{F}$ that has at least one unscanned crossing.
   (b) Select the first unscanned crossing c on p_f, and let p_b in $\mathcal{B}$ correspond to a backward part containing c.
   (c) Recombine p_f and p_b at c. Label p_f and all previous crossings on p_b as scanned. If p_b was already recombined with some other forward part p'_f, mark p'_f as unscanned.
   (d) Are there any unscanned forward parts with unscanned crossings left?
   YES: Go to step 4a.
   NO: Stop: a solution is found that is vertex-disjoint.

5.2 Minimum cut sizes

We now give the approximation algorithm for finding the minimum cut between two vertices s and t. Assume again we have a ToR graph $G = (V, E)$ and two vertices $s, t \in V$. We also assume there is no direct edge in G between s and t, since a s-t cut does not exist in that case. The algorithm is then as follows:

2-Approximation algorithm MinCut (Erlebach et al., 2005)

1. Construct the 2-layer graph, and calculate a minimum cut in this graph.
2. From the cut found in step 1, construct a cut $\mathcal{C}$ in G as follows: $\mathcal{C}$ contains all vertices $v \in V$ for which at least one copy is in the cut found in step 1.
3. Stop: $\mathcal{C}$ is a cut in G containing at most twice the number of nodes as in a minimum cut.

6. Computational experiments

In this section we first give a description of the data we used for our experiments (Section 6.1). Next, in Section 6.2, we discuss some issues concerning the implementation of the algorithms, and finally we present our results. In Section 6.3 we give computational results and discuss the performance of the different algorithms. The algorithms described in Sections 3, 4 and 5 are executed on a number of different ToR graphs, and we compare these results with those in the undirected model (where routing policies that are consequences of established economic relationships are not included) in order to quantify what the differences are with respect to the size of a minimum cut and the number of disjoint paths. Finally, in Section 6.4, we focus on the interpretation of the results.

6.1 Description of the data

We use BGP tables from five different dates (April 2001, February 2002, April 2002, January 2003 and February 2004), available from the University of Oregon Route Views project web-site (OREGON), to construct undirected graphs and four types of ToR graphs. This means that we have five different graphs for each of the five points in time, giving five undirected graphs and 20 ToR graphs in total. The undirected graphs are obtained by creating an undirected edge between two ASs if they appear consecutively in some path in the BGP tables. We also used one undirected graph model representing the Internet of April 1–16, 2002 that we obtained from CAIDA's Internet Topology Data Kit, ITDK0204 (CAIDA). We refer to this graph as the *CAIDA graph*, to the undirected graphs based on Oregon Route Views data as *undirected BGP graphs*, and to the graphs that include AS relationships as *ToR graphs*. The types of ToR graphs are denoted by A, B, C, and D as follows:

- ToR graphs of type A are obtained using the algorithm from Erlebach et al. (2002). They contain only customer-provider edges, no peer-to-peer or sibling edges.
- ToR graphs of type B are obtained using the algorithm from Di Battista et al. (2003) by running the software bgpSat publicly available from their web-page (BGPSAT). A majority of the edges are classified by bgpSat as customer-provider edges, but the classification of some edges is left undetermined. We classify the latter edges as peer-to-peer edges. Thus, type B graphs contain customer-provider edges and a few peer-to-peer edges.
- ToR graphs of type C are obtained from the web-page (CIMVP) and have been produced with the algorithm from Subramanian et al.

(2002). The algorithm classifies edges as peer-to-peer edges, customer-provider edges, or unknown edges. We treat the unknown edges as sibling edges.

- ToR graphs of type D are obtained with the algorithm from Gao (2001) (using the implementation (LRIP)) and contain customer-provider edges, peer-to-peer edges, and sibling edges.

Table 1: Comparison of edge classifications.

ToR Graphs	*Percentages of identically classified edges*				
	18.04.2001	*04.02.2002*	*06.04.2002*	*09.01.2003*	*10.02.2004*
A vs. B	95.53	95.41	95.40	95.88	95.08
A vs. C	91.70	91.57	92.21	92.24	91.02
A vs. D	90.96	91.43	91.67	93.16	91.23
B vs. C	89.71	90.30	90.55	90.40	90.28
B vs. D	89.37	90.46	90.59	91.50	90.24
C vs. D	89.60	90.55	90.72	91.35	90.75

All of the inference algorithms that we have used for the construction of ToR graphs are heuristics. Thus, it is interesting to see how many edges between ASs are classified in the same way by the different algorithms. In Table 1 the percentages of identically classified edges are given for all six combinations of ToR graphs. For example, 95.53% of the edges are classified in the same way in A and B graphs from April 2001, as is shown in the first entry of the table. From this table we see that approximately 90% of all edges are classified the same.

Since computing the maximum number of vertex-disjoint paths and the minimum cut size for *all* pairs of ASs would have taken prohibitively long (even after pruning vertices of degree 1, the graphs still contain roughly 7,000 to 11,000 vertices), we confine our calculations to approximately 1000 pairs of ASs per graph. For this reason, we select 47 ASs as representatives and carry out the computations for all possible 1081 pairs of these ASs. We have selected the ASs by taking 47 vertices among the vertices of largest degree in the biggest R component of the undirected BGP graph of April 2002. (A partition of Internet graphs into P, Q, R and I components was proposed by Vukandinović et al. (2002). The biggest R component is the biggest connected component in the graph that is obtained after deleting all vertices of degree 1 and their neighbors.) All of the 47 selected ASs are vertices in that component that have at least 7 neighbors within that component. Their AS numbers and descriptions are given in Table 2. As one can see, the ASs are geographically well spread—they are from Europe, USA, and Asia. Furthermore, there are representatives of bigger and smaller ISPs (Internet Service Providers), telecom nodes (e.g. Japanese and Belgian telecom), well-

connected universities and research centers (e.g. University of Stanford, University of Oregon, and National Center for Supercomputing Applications), exchange points (e.g. London and Hongkong Internet Exchange), etc. This means that we have chosen well-connected ASs with diverse functionalities and good geographic coverage while avoiding the highest-degree nodes in the Internet (which are neighbors of leaves) as well as nodes with very small degree.

6.2 Implementation issues

We have implemented the algorithms in C++ using CPLEX 9.0 to solve linear programs and the LEDA library to process graphs. Our experiments were done on a Sun Fire 480R workstation with two 900MHz processors (our code uses only one of them) and 4GB main memory.

For all computations we have removed vertices with degree 1, since they do not affect the number of disjoint paths or the cut sizes for any other pair of ASs. After pruning the leaf vertices, the graphs contain about 7,000 to 11,000 vertices and 20,000 to 30,000 edges.

For the computation of disjoint paths and cuts, we replaced each peer-to-peer edge $\{u, v\}$ by two edges (u, d) and (d, v), where d is a new dummy vertex. In this way the valley-free path policy is preserved, while the graph consists of directed edges only.

At the start of the branch-and-price algorithm for computing the maximum number of disjoint s-t paths, we do some additional preprocessing on the graphs. First, we delete all vertices (except s and t) for which the in-degree is equal to zero. These vertices can never belong to a valid path, so removing them will not affect the solution we find. Next, we check whether s and t belong to the same biconnected component of the underlying undirected graph. (A biconnected component of an undirected graph G is a subgraph of G such that we can remove any vertex of this subgraph without disconnecting it (Harary, 1969).) If so, we can run the algorithm on this component only (which is usually much smaller than the original graph), and in this way we still get the optimal solution. If s and t do not belong to the same biconnected component, the number of valid paths between s and t will be either 0 or 1. So we check whether there exists a valid path from s to t, in which case the number of vertex-disjoint paths is equal to one. If no valid s-t path exists, our solution is equal to zero. Finally, we found that adding the valid inequalities discussed in Section 3.1.3 actually slows down the branch-and-price algorithm. In fact, the number of branching nodes needed to solve the problem decreases, as expected, but the time needed to process a single node increases more heavily than the decrease in number of branching nodes, so the computational results presented next are those obtained without the additional valid inequalities.

For the minimum cut problem, we need to get a well-defined notion of

Table 2: The 47 selected ASs with AS numbers and descriptions from Internet registries.

AS nr.	*Description*
32	Stanford University
237	Merit Network Inc.(USA)
600	OARnet(USA)
680	DFN-IP service G-WiN
1136	KPN Telecom OVN IO
1237	Korea Institute of Science and Technology Information
2500	Japan Network Information Center WIDE Project
2514	NTT PC Communications, Inc., Japan
2518	C&C Internet Service mesh(NEC Corporation), Japan
2647	SITA France
2687	IBM, NH USA
2818	BBC Internet Services, UK
3112	OARnet(USA)
3304	KPN Belgium
3333	RIPE NCC Operations
3491	CAIS Internet(USA)
3557	Internet Systems Consortium, Inc. (USA)
3582	University of Oregon
3754	NYSERNet(USA)
4197	ERX-GLOBALONLINE, Japan
4635	Hong Kong Internet Exchange–Route Server 1
4725	ODN JAPAN TELECOM CO.,LTD.
5000	Internet Online Services (USA)
5056	Iowa Network Services (USA)
5413	GX Networks
5459	LINX-AS,London Internet Exchange Ltd.
6079	RCN Corporation (USA)
6402	One Call Communications, Inc.(USA)
6774	BELBONE BELGACOM
6830	UPC Distribution Services european broadband ISP services
7091	ViaNet Communications (USA)
7623	HPCNET-AS High Performance Computing NETwork(HPCNET)Korea
7660	APAN-JP Asia Pacific Advanced Network - Japan
7679	QTNET Kyushu Telecommunication Network Co.,Inc.
8426	CLARANET-AS ClaraNET UK AS of European ISP
8553	AVENSYS Avensys Networks Ltd UK
9270	APAN-KR-AS Asia Pacific Adv. Netw. Korea Consort. Net. Oper.Center
9335	CIP-JAPAN-AS-AP ATT IPlus Asia and Pacific IP Network
9497	DIGITELONE Digital Telecommunications Philippines Inc.
10099	HKUNICOM1-AP Voice over IP, ISP
10764	National Center for Supercomputing Applications
11854	Internap Network Services (USA)
12359	INTELIDEAS Intelideas Autonomous System Madrid, Spain
12457	ONO-SERVICE-PROVIDER, Spain
13129	Global Access Telecommunications, Inc.
13646	Cignal Global Communications, Inc. (USA)
14390	Core Communications, Inc (USA)

minimum s-t cuts also for adjacent vertices. We handle such vertex pairs as follows: we remove the direct edge (or pair of edges, in the case of a sibling relationship between s and t) between the two vertices s and t, compute the size of a minimum s-t cut in the graph without that edge, and add 1 to the result. We do this in the undirected graphs as well as in the ToR graphs. Note that in the undirected model, the number of disjoint paths between two vertices is equal to the size of a minimum cut separating these two vertices. In ToR graphs, these values can differ.

6.3 Computational results

Next, we are interested in the number of disjoint paths and the minimum cut size between pairs of ASs in the different graphs. First we discuss the performance of the two exact algorithms for solving the vertex-disjoint paths problem. Then we give results for the performance of the algorithm for solving the minimum cut problem, and finally we give results from the approximation algorithms.

6.3.1 Vertex-disjoint paths

We have tested both the branch-and-price and the branch-and-bound algorithm described in Section 3 to calculate the maximum number of vertex-disjoint paths for any pair of ASs. In Tables 3 and 4 we give the results of these computations.

Tables 3 and 4 give the computational results for both algorithms. The first two columns show the graph type and date. The third column contains the value of the integer optimum. The last four columns show the computation times (in seconds), the number of branching nodes needed to solve the problems, the percentage of problem instances that are solved in less than one second, and the percentage of instances that are solved in more than 10 seconds (Table 3 contains these values for the branch-and-price algorithm; Table 4 for the branch-and-bound algorithm). All values in these tables are average values over the 1081 pairs of ASs, so they contain results of over 20,000 problem instances. While we could run the branch-and-price algorithm to completion on all pairs in all graphs, we had to terminate the branch-and-bound algorithm on a few pairs (at most 10 out of 1081 pairs in each of the graphs) after several hours of computation time. The running-time and the number of branching nodes shown for the branch-and-bound algorithm in Table 4 are thus the averages over the pairs for which the algorithm could be run to completion.

From Tables 3 and 4 we conclude that, on the average, both algorithms perform well on the selected pairs of ASs. The running-times of the branch-and-bound algorithm are much more variable. On 71.78% of all instances,

Table 3: Results for branch-and-price algorithm.

Type	*Date*	*Opt*	*Branch&Price*			
			Time	*#BN*	*%≤1*	*%>10*
A	18.04.2001	6.38	0.83	1.96	98.06	0.65
	04.02.2002	7.88	1.26	2.02	94.36	1.30
	06.04.2002	8.66	2.61	4.55	93.15	1.85
	09.01.2003	7.35	0.80	1.58	94.91	0.65
	10.02.2004	8.10	6.26	6.77	86.12	3.79
B	18.04.2001	6.42	3.34	7.59	91.77	3.15
	04.02.2002	8.49	7.61	11.21	81.41	5.46
	06.04.2002	9.39	10.69	10.94	78.08	7.77
	09.01.2003	7.52	2.82	5.06	86.40	3.61
	10.02.2004	8.44	12.12	10.90	79.19	5.64
C	18.04.2001	6.14	1.25	2.29	94.54	1.30
	04.02.2002	7.98	2.04	2.76	86.86	2.68
	06.04.2002	8.46	2.96	3.71	86.77	2.41
	09.01.2003	6.61	0.88	1.57	93.06	0.83
	10.02.2004	7.80	1.94	1.97	80.48	2.50
D	18.04.2001	6.34	2.63	3.06	83.63	4.26
	04.02.2002	8.01	2.20	2.42	85.38	3.70
	06.04.2002	8.69	5.96	4.31	81.41	3.98
	09.01.2003	7.30	1.80	2.20	88.53	2.41
	10.02.2004	7.92	8.36	4.16	73.64	4.81

the branch-and-bound algorithm was faster than the branch-and-price algorithm. On the other hand, the branch-and-price algorithm could solve all instances in reasonable time (the average running-time over all instances is 3.92 seconds, while the instance with the longest running-time took slightly more than one hour), while the running-time of the branch-and-bound algorithm increased drastically for a few instances, thus leading to a larger average running-time on most graphs. The number of branching nodes needed to find the integer optimum is much larger for the branch-and-bound algorithm in comparison to the branch-and-price algorithm. For the branch-and-price algorithm, the average number of branching nodes is surprisingly small, since in about 89% of the problem instances the solution to the LP-relaxation is integral and we do not need to branch at all.

6.3.2 Minimum cuts

The algorithm described in Section 4 to calculate the size of a minimum cut for a pair of ASs has also been executed on the ToR graphs. The results of these computations can be found in Table 5.

Table 5 gives the results for the ToR graphs. The first two columns show the graph type and the date, the third column contains the optimal value of the minimum cuts, and finally we give the computation times (in seconds),

Table 4: Results for branch-and-bound algorithm.

Type	*Date*	*Opt*	*Branch&Bound*			
			Time	*#BN*	*%≤1*	*%>10*
A	18.04.2001	6.38	9.44	34.42	94.26	1.11
	04.02.2002	7.88	2.63	10.84	88.99	2.59
	06.04.2002	8.66	4.40	16.25	87.60	3.61
	09.01.2003	7.35	2.28	5.31	94.63	1.85
	10.02.2004	8.10	15.40	32.86	86.96	4.44
B	18.04.2001	6.42	2.16	10.45	83.44	3.33
	04.02.2002	8.49	25.62	71.69	71.14	8.33
	06.04.2002	9.39	42.63	115.76	68.55	12.86
	09.01.2003	7.52	11.62	33.28	82.79	3.05
	10.02.2004	8.44	18.13	40.19	72.34	8.97
C	18.04.2001	6.14	3.81	11.02	86.12	4.53
	04.02.2002	7.98	18.50	43.59	69.29	6.94
	06.04.2002	8.46	4.28	10.68	69.47	3.33
	09.01.2003	6.61	1.73	4.38	84.55	2.59
	10.02.2004	7.80	70.27	133.88	71.14	10.73
D	18.04.2001	6.34	30.43	67.72	71.51	9.44
	04.02.2002	8.01	47.57	116.08	73.64	9.90
	06.04.2002	8.69	45.04	88.61	58.19	13.97
	09.01.2003	7.30	14.20	31.43	79.56	5.00
	10.02.2004	7.92	59.74	79.00	66.51	16.37

the number of branching nodes needed to find the integer optimum, the percentage of problem instances that are solved in less than one second, and the percentage of instances that are solved in more than 10 seconds. Again, all values are average values over all 1081 pairs of ASs for a specific graph type and date.

As can be seen from Table 5, the algorithm for finding the minimum cut sizes in ToR graphs is very fast, also compared to the computation times for the algorithms for finding the maximum number of vertex-disjoint paths. Again, the number of branching nodes needed to find the integer optimum is small, since the solution to the LP-relaxation is integral in 98.5% of the problem instances.

6.3.3 Approximation algorithms

In Table 6 we give results for the 2-approximation algorithms presented in Section 5. In the first two columns we show the graph types and the different dates. Columns three to five contain information on the number of vertex-disjoint paths, namely the optimal value, the value found by the approximation algorithm and the computation times, and in the last three columns we give the same results for the sizes of minimum cuts. Again, all values are average values over all 1081 problem instances for a specific graph

Table 5: Results for the minimum cut problem in ToR graphs.

Type	*Date*	*Opt*	*Time*	*#BN*	*%≤1*	*%>10*
A	18.04.2001	6.38	0.69	1.03	94.73	0.09
	04.02.2002	7.88	0.95	1.01	77.15	0.00
	06.04.2002	8.66	1.04	1.02	67.07	0.09
	09.01.2003	7.35	1.01	1.01	70.68	0.19
	10.02.2004	8.11	1.90	1.06	49.68	0.74
B	18.04.2001	6.43	0.82	1.11	89.18	0.46
	04.02.2002	8.52	1.86	1.33	59.85	2.41
	06.04.2002	9.42	1.85	1.16	47.92	2.22
	09.01.2003	7.53	1.23	1.03	60.13	0.09
	10.02.2004	8.44	1.83	1.05	41.81	1.11
C	18.04.2001	6.15	1.02	1.06	81.22	0.37
	04.02.2002	7.99	1.43	1.07	60.22	0.46
	06.04.2002	8.47	1.58	1.12	57.45	0.93
	09.01.2003	6.61	1.14	1.02	65.22	0.09
	10.02.2004	7.81	2.15	1.14	34.97	1.39
D	18.04.2001	6.35	1.26	1.17	71.42	0.74
	04.02.2002	8.02	1.34	1.05	63.74	0.28
	06.04.2002	8.70	3.04	1.32	49.58	1.02
	09.01.2003	7.30	1.22	1.01	50.32	0.09
	10.02.2004	7.93	2.08	1.11	27.10	1.76

type and date.

From these results we conclude that both approximation algorithms perform really well. For the problem of finding the maximum number of disjoint paths we find that in 41.14% of all instances we get an optimal solution, and for 67.63% the difference between the optimal value and the value found by the approximation algorithm is at most 1. For the problem of finding the minimum cut sizes, 90.82% of the instances are solved optimally, and in 97.63% of the instances the difference between the optimum and the value of the approximation algorithm is at most 1. So, the heuristic for finding the minimum cut sizes is extremely well suited for this type of instances. The computation times for both algorithms are really fast: all problem instances for both problems are solved within less than one second of computation time.

6.4 Interpretation of the results

In this section we describe how the results that we obtained can be interpreted. First we discuss the connectivity of the Internet as measured by the number of disjoint paths and minimum cut sizes. Then, in all four types of ToR graphs we also compute the number of edges that are contained in directed customer-provider cycles as well as the fraction of pairs of ASs that are connected with directed customer-provider paths in order to gain more

Table 6: Results for approximation algorithms.

Type	*Date*	*Disjoint paths*			*Cut sizes*		
		Opt	*Approx*	*Time*	*Opt*	*Approx*	*Time*
A	18.04.2001	6.38	5.32	0.18	6.38	6.40	0.19
	04.02.2002	7.88	6.61	0.23	7.88	8.03	0.24
	06.04.2002	8.66	7.23	0.24	8.66	8.84	0.25
	09.01.2003	7.35	6.36	0.26	7.35	7.41	0.28
	10.02.2004	8.10	6.86	0.32	8.11	8.23	0.34
B	18.04.2001	6.42	5.28	0.18	6.43	6.53	0.19
	04.02.2002	8.49	6.82	0.24	8.52	8.70	0.25
	06.04.2002	9.39	7.45	0.26	9.42	9.57	0.27
	09.01.2003	7.52	6.18	0.28	7.53	7.66	0.30
	10.02.2004	8.44	6.93	0.35	8.44	8.66	0.37
C	18.04.2001	6.14	5.18	0.22	6.15	6.19	0.23
	04.02.2002	7.98	6.70	0.27	7.99	8.14	0.28
	06.04.2002	8.46	7.08	0.28	8.47	8.64	0.29
	09.01.2003	6.61	5.79	0.29	6.61	6.70	0.31
	10.02.2004	7.80	6.71	0.37	7.81	8.01	0.41
D	18.04.2001	6.34	5.19	0.22	6.35	6.49	0.23
	04.02.2002	8.01	6.57	0.27	8.02	8.09	0.27
	06.04.2002	8.69	7.20	0.27	8.70	8.97	0.28
	09.01.2003	7.30	6.24	0.28	7.30	7.37	0.30
	10.02.2004	7.92	6.74	0.35	7.93	8.08	0.38

insight into the AS hierarchy produced by the different inference algorithms.

6.4.1 Connectivity measures for the Internet

In Table 7 we compare the number of vertex-disjoint paths for the different types of ToR graphs, the undirected BGP graphs and the CAIDA graph. In the second column we give the average number of vertex-disjoint paths, averaged over all pairs of ASs and all dates of the specified graph type. The third column gives the minimum number of paths found, and the last column shows the maximum number of vertex-disjoint paths (the CAIDA graph is available only for one date, and 3 of our 47 selected ASs are missing from that graph; AS pairs involving a missing AS node were thus ignored for the CAIDA graph).

In Table 8 we compare the size of the minimum cuts in ToR graphs with results from the undirected models, and for all graph types we give the average over all pairs and dates, the minimum value of a minimum cut, and the maximum value. For the undirected BGP graphs and the CAIDA graph, these values are the same as for the vertex-disjoint paths problem, since the max-flow min-cut equality holds for the undirected graphs.

If we compare the connectivity of the ToR graphs with the undirected

Table 7: Vertex-disjoint paths in ToR and undirected graphs.

Graph Type	*avg VDP*	*min VDP*	*max VDP*
A	7.67	1	55
B	8.05	1	65
C	7.40	0	60
D	7.65	1	48
undirected BGP	13.46	2	107
CAIDA	12.74	6	108

Table 8: Minimum cut sizes in ToR and undirected graphs.

Graph Type	*avg CS*	*min CS*	*max CS*
A	7.68	1	56
B	8.07	1	65
C	7.40	0	60
D	7.66	1	48
undirected BGP	13.46	2	107
CAIDA	12.74	6	108

models, we see a big difference (see Tables 7 and 8). The number of disjoint paths, and the cut sizes, are much larger in the undirected models. For about 72% of all pairs, the number of disjoint paths (and the minimum cut size) is at least 1.5 times bigger in the undirected models, as compared to the ToR graphs, and for approximately 44%, these values in the undirected models are at least twice as large than in the ToR graphs.

When we look at the differences in connectivity between the four different ToR graphs we see that there is no striking difference between the number of disjoint paths and the sizes of minimum cuts. Generally speaking, graphs of type B have the highest connectivity and graphs of type C have the lowest connectivity (see third column of Tables 3, 4, and 5). However, the connectivity of the different ToR graphs seems to be similar.

In Figure 6, the four types of ToR graphs are represented together with the undirected BGP graph and the CAIDA graph, all graphs taken from April 2002. We obtained similar results for the other four dates, but since we had the CAIDA graph only for April 2002, we chose to use this date for the illustration. The number of disjoint paths and the minimum cut size are shown for each of the 1081 AS pairs in all six graphs. The values are sorted in order of non-decreasing values in the undirected BGP graph. As the figure shows, there is no striking difference among the ToR graphs. The values for the undirected BGP graph, however, are significantly higher than those for the ToR graphs. This clear difference between the undirected and ToR models indicates that, in order to get an accurate picture of the Internet structure and connectivity, it is important to take routing policies into account.

The values for the CAIDA graph, which has about 6% more edges than the undirected BGP graph, are somewhat incomparable to those of the undirected BGP graph. For about 35% of the AS pairs, the CAIDA graph has more disjoint paths (up to 100 more paths for one pair), and for about 59% of the pairs, the undirected BGP graph has more disjoint paths (up to 69 more paths for one pair). This indicates that some parts of the Internet are denser (higher number of edges) in the CAIDA graph, while other parts are denser in the undirected BGP graph.

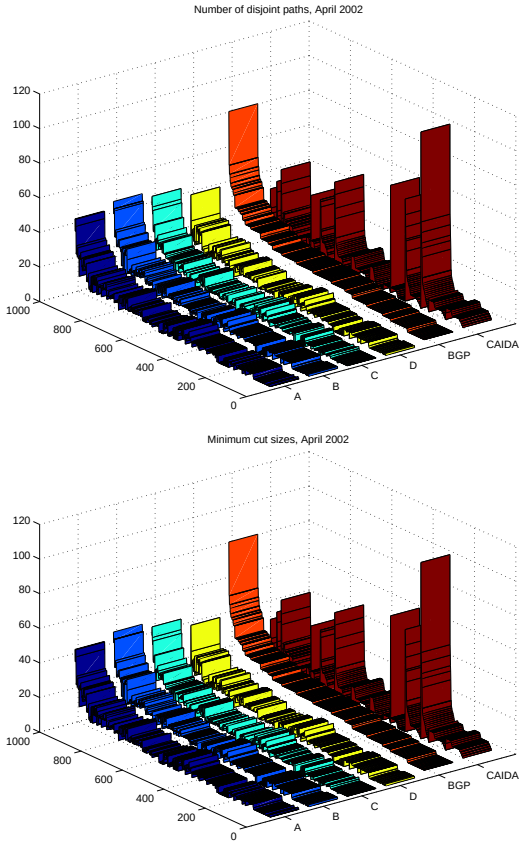

Figure 6: Comparison of ToR and undirected graphs.

Let us now discuss trends over time. The trends for the number of disjoint paths between the different time periods are shown in Figure 7 for each of the four types of ToR graphs and for the undirected BGP graphs. There are four plots, each of them corresponding to a particular time period. In each plot, there is a bar for each of the five graph types. The white part of the bar represents the number of pairs of ASs for which the number of disjoint paths increased in this time period; the shaded part of the bar corresponds to the number of pairs for which the number of disjoint path

stayed the same, and the black part is the number of pairs for which the number of paths decreased in this time period. The results for the minimum cut sizes are similar, so we omit them here.

The figure shows that the ToR graphs behave similarly for all time periods. In the first two time periods, the AS pairs with increasing connectivity form the majority. Then, in the third time period, more than half of the AS pairs display decreasing connectivity. Finally, in the fourth time period, the ToR graphs have roughly the same number of AS pairs with increasing and decreasing connectivity, respectively, while about 70% of the AS pairs display increasing connectivity in the undirected BGP graphs.

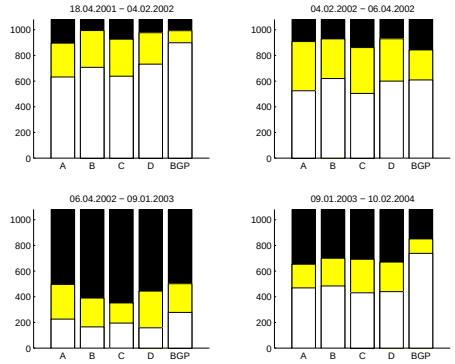

Figure 7: Trends over time for ToR and undirected BGP graphs.

When we study the differences between the number of disjoint paths and the cut size in ToR graphs, we find that these numbers are equal for about 99% of all AS pairs in each of the ToR graphs (see third column of Tables 3, 4, and 5). The absolute difference between the minimum cut size and the number of disjoint paths was never larger than 2 for any of the AS pairs in any of the ToR graphs. Thus, the minimum cut size does not differ significantly from the maximum number of disjoint valid paths in our ToR graphs. Notice that this difference could be as large as a *factor* of 2 in general graphs (Erlebach et al., 2005).

6.4.2 Directed customer-provider cycles

We call a directed cycle (as defined in Section 2) in a ToR graph a *customer-provider cycle* if it contains only customer-provider edges. If the Internet was a strictly hierarchical network (i.e., if levels can be assigned to the ASs in such a way that, in any customer-provider relationship, the customer is on a lower level than the provider), one would expect that there are no customer-provider cycles in ToR graphs at all. Therefore, one might use the existence of such cycles as a sign of a misclassification.

We check the existence of such cycles in each of the ToR graphs as

Table 9: Results for directed customer-provider cycles.

Type	*Date*	*Total*	*Min*	*Max*
A	18.04.2001	2571	3	9
	04.02.2002	2441	3	9
	06.04.2002	2278	3	10
	09.01.2003	2182	3	10
	10.02.2004	3453	3	8
B	18.04.2001	4046	3	8
	04.02.2002	4710	3	8
	06.04.2002	4825	3	9
	09.01.2003	4858	3	9
	10.02.2004	6802	3	9
D	18.04.2001	318	3	20
	04.02.2002	16	3	5
	06.04.2002	9	3	3
	09.01.2003	69	3	11
	10.02.2004	428	3	14

follows. First, we remove all sibling edges and peer-to-peer edges from the graph. Then, for each customer-provider edge from AS_i to AS_j, we calculate a shortest directed path (i.e, a path with the smallest number of edges) from AS_j to AS_i. Such a path exists if and only if the edge from AS_i to AS_j is contained in at least one directed cycle. If such a path is found, it gives us a shortest customer-provider cycle containing the edge.

We find that there are no customer-provider cycles in the ToR graphs of type C, except in the graph for 09.01.2003; for the latter date, the type C graph contains a single customer-provider cycle with four nodes (AS11563, AS19035, AS17819, AS1668). In Table 9, we give the results that we obtained for ToR graphs of type A, B and D. For each of the graphs, we show the total number of customer-provider edges that are contained in cycles, the minimum length of the shortest cycle containing a customer-provider edge, and the maximum length of the shortest cycle containing a customer-provider edge. We find that type B graphs have the largest number of customer-provider cycles, type A graphs have about half as many, and type D graphs have much fewer cycles than both A and B graphs.

As the ToR graphs of type A and B contain no sibling edges and either no or very few peer-to-peer edges, a larger number of customer-provider cycles could be expected in these graphs. Table 9 confirms that significantly more edges are contained in customer-provider cycles in these graphs. Most of the cycles in the graphs of type A and B are caused by edges classified as customer-provider in A or B graphs, but classified in D graphs as peer-to-peer, sibling or provider-customer edges.

In the A graph from 18.04.2001, there are 2571 edges contained in cycles.

Each of these edges is contained in a shortest cycle. Among these 2571 shortest customer-provider cycles (we consider a cycle multiple times if it is the shortest customer-provider cycle of several edges), 1909 have an edge classified as peer-to-peer in the corresponding D graph, 574 of the remaining ones have an edge classified as sibling edge in the D graph, and 67 of the remaining ones have an edge classified as provider-customer edge in D. Only 21 of the 2571 cycles are also present in the D graph. Qualitatively similar results are obtained for all dates for the A and B graphs.

Analyzing the directed cycles in the D graphs, we found that all customer-provider cycles can be eliminated by deleting a very small number of edges (12, 4, 3, 8, and 11 edges, respectively, in the five D graphs from 18.04.2001 to 10.02.2004).

We checked manually 10 edges that were contained in more than 50 discovered cycles (up to 348 cycles) in the D graph from 10.02.2004, using the Nemecis tool (NEMECIS) to access data from Internet Routing Registries. For three of these edges there was no information in the Internet Registries, 6 of them were classified as peer-to-peer edges (i.e., at least one of the two ASs registered this particular edge as peer-to-peer) and only one edge was registered as customer-provider (confirming its classification in the D graph).

Although it is still possible that some of the directed customer-provider cycles are not caused by misclassifications, we think that they are a good starting point for the detection of misclassifications, in particular if their analysis is combined with a comparison between the different ToR graphs and checking of entries in Internet Registries. Such cycles could be used to introduce peer-to-peer edges and sibling edges into the ToR graphs of type A and B, which contain essentially only customer-provider edges (in our type B graphs, the only peer-to-peer edges are those that were left unclassified by the algorithm from Di Battista et al. (2003)).

6.4.3 The depth of the provider hierarchy in ToR graphs

Finally, in order to examine the typical nature of AS paths in the different ToR graphs, we investigated how many pairs of vertices can be connected by directed paths, i.e., by paths going "only up" or "only down." A path $AS_1, \ldots, AS_k$ between two ASs AS_1 and AS_k such that each AS_i is a customer of AS_{i-1}, for $2 \leq i \leq k$, is called a *customer chain*. In our experiments, we check for all pairs of ASs in ToR graphs (except the pairs involving leaf vertices) whether one of the two ASs is connected to the other via a customer chain. For such pairs of vertices, it is possible to use paths only through customers (at least in one of the two directions) and thus take advantage of the "customer-preference" policy. Namely, routing through a customer brings profit, through a peer is neutral, and through a provider incurs costs for the sender (Spring et al., 2003).

The statistics about customer chains in all four types of ToR graphs are

Table 10: Percentage of pairs of ASs connected by customer chains.

Date	*A*	*B*	*C*	*D*
18.04.2001	10.01%	14.93%	0.52%	2.25%
04.02.2002	7.52%	14.03%	0.56%	0.67%
06.04.2002	7.02%	14.05%	0.53%	0.59%
09.01.2003	6.84%	13.62%	0.47%	0.84%
10.02.2004	7.60%	14.65%	0.53%	1.42%

given in Table 10. This table shows for each of the five dates the percentages of pairs of ASs that are connected by customer chains in all our types of ToR graphs.

About 6–10% of all pairs in type A graphs and 13–15% of all pairs in type B graphs are connected by customer chains. For graphs of type C and D the number is significantly smaller, which was to be expected because they contain substantially more edges that are not customer-provider edges. This indicates that in graphs of type A and B, the hierarchy seems to be similar and tends to be deep. In type C and D graphs, the hierarchy seems to be wider, as there are many more pairs that are connected only through paths going "up and then down."

7. Conclusions

We have compared different types of graphs with inferred AS relationships (ToR graphs) regarding connectivity measures and path characteristics. We have studied the maximum number of disjoint valid paths and the minimum cut size for selected AS pairs. Since both problems are $\mathcal{NP}$-hard, we have designed and implemented several algorithms that allowed us to compute optimal values for all pairs among a set of representative ASs. For the problem of finding the maximum number of disjoint paths between any pair of ASs, we have implemented two exact algorithms, the first one being a branch-and-price algorithm based on an integer programming formulation of the problem, and the second one being a branch-and-bound algorithm in which we perform a max-flow calculation in each node of the search tree. From the results we conclude that the latter algorithm is often faster than the first but may require excessive computation times on certain inputs, while the computation times for the branch-and-price algorithm are always acceptable and do not display such a variability. For the problem of finding the minimum cut sizes, we have implemented a branch-and-cut algorithm that performs really well, with average computation times around one to two seconds for instances with up to 11,000 nodes and 30,000 edges.

The results of these algorithms allow us to quantify the differences in connectivity between ToR graphs and the traditional undirected model of

the Internet, which ignores routing policies. We find that about 44% of the selected AS pairs have more than twice as many disjoint paths in the undirected model than in the ToR graphs, which implies that the use of ToR graphs is crucial for Internet analysis and simulations that are sensitive to connectivity properties, e.g. in studies concerning topological robustness, multi-path routing, etc. We have also investigated the increase of connectivity over time and found that the number of disjoint paths between ASs seems to grow for fewer AS pairs in the ToR graphs than in the undirected graph model.

Comparing the ToR graphs with each other, we find that on the average they do not differ much with respect to the number of disjoint paths and the minimum cut sizes between AS pairs. On the other hand, concerning the hierarchy (observed indirectly by counting the number of AS pairs connected through customer chains) it turns out that A and B graphs are relatively similar to each other, but different from C and D graphs—their hierarchy appears to be deeper than that of C and D graphs. In addition, we find that the investigation of short directed customer-provider cycles in the ToR graphs can help to detect misclassifications and may lead to new approaches for introducing peer-to-peer or sibling relationships into A and B graphs, which can make these models more realistic.

While our investigations provide some insight into the properties of the ToR graphs produced by the different available inference algorithms, it is not possible for us to identify one of these algorithms as better than the others. Researchers who employ ToR graphs in their research should be aware of the differences in the ToR models produced by different algorithms and make sure that their conclusions are not biased by the choice of ToR graph. Our findings can help in making informed decisions about the choice of a ToR graph model.

Furthermore, our approach of adapting the classical connectivity measures, maximum number of disjoint paths and minimum cut size, to valley-free paths in ToR graphs can be useful in further research on robustness issues in the Internet. Besides, it may be possible to adapt our branch-and-price approach to incorporate other types of constraints on valid paths, thus allowing the analysis of connectivity properties of other networks with special routing constraints as well.

The known algorithms for inferring AS relationships from Gao (2001, Subramanian et al. (2002), Di Battista et al. (2003), and Erlebach et al. (2002) all need data from BGP routing tables as input. As the data from BGP routing tables is not always complete or accurate (the impact of this is demonstrated convincingly by the huge difference in the number of disjoint paths for certain AS pairs in the undirected BGP graph and the CAIDA graph, see Figure 6), it would be an interesting question for future research whether good inference of AS relationships is also possible without knowledge of BGP routing tables. Such an inference algorithm

could then also be used for classifying AS relationships in more complete undirected AS graphs (such as the union of the undirected BGP graph and the CAIDA graph) or in synthetic graph models obtained from Internet topology generators. A different approach in the latter direction has been explored by Chang et al. (2003), where a new optimization-driven model for Internet growth is presented that allows the generation of synthetic AS graphs containing only customer-provider relationships.

Finally, let us summarize our findings by answering the research question posed in Section 1.

- The undirected graph model of the Internet topology is not suited for studying stability issues of the Internet. Of course, any graph-theoretical model of the Internet is an approximation of reality. However, some approximations are better than others: our results show that from the viewpoint of connectivity, using the undirected graph model may lead to serious misjudging of the connectivity.

- The different heuristics (Gao, 2001; Subramanian et al., 2002; Di Battista et al., 2003; Erlebach et al., 2002) used for constructing a topology do not differ much with respect to the connectivity measures. On the average, they all have a similar number of disjoint paths and minimum cut size between pairs of ASs.

- Graphs of type A and B do not have sibling edges and no or very few peer-to-peer edges, and this causes the existence of a relatively large number of customer-provider cycles. Graphs of type D contain a significant number of peer-to-peer and sibling edges, and they have few customer-provider cycles. Graphs of type C contain no customer-provider cycles at all (except for a single cycle of length 4 for one date). Depending on the question one wants to investigate, this could be relevant. The directed customer-provider cycles in ToR graphs can be useful for the detection of misclassified edges, especially if the analysis is combined with a comparison between the different ToR graphs.

- We obtain optimal solutions to the problems of finding the maximum number of vertex-disjoint paths and minimum cut sizes, using the exact algorithms proposed in this chapter. The algorithms require, on average, a small amount of time to find these optimal values. However, for a small number of instances, the branch-and-bound algorithm was unable to find an optimal solution.

- The performance of the approximation algorithms is reasonable. Especially for the problem of finding minimum cut sizes, the approximation algorithm performs really well. Both these algorithms are much faster than the exact algorithms.

References

Ahuja, A., Magnanti, T., and Orlin, J., editors (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, N.J.

Akella, A., Chawla, S., Kannan, A., and Seshan, S. (2003a). Scaling properties of the internet graph. In *ACM PODC '03*.

Akella, A., Maggs, B., Seshan, S., Shaikh, A., and Sitaraman, R. (2003b). A measurement-based analysis of multihoming. In *SIG-COMM '03*.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

BGPSAT. G. Di Battista, M. Patrignani and M. Pizzonia. Computing the types of the relationships between Autonomous Systems. Project webpage: http://www.dia.uniroma3.it/∼compunet/relationships/.

CAIDA, The Cooperative Association for Internet Data Analysis. Internet Topology Data Kit #0204 (Data collected as part of CAIDA's skitter initiative). http://www.caida.org/tools/measurement/skitter/. Support for skitter is provided by DARPA, NSF, and CAIDA sponsorship.

Chang, H., Govidan, R., Jamin, S., Shenker, S., and Willinger, W. (2004). Towards capturing representative AS-level Internet topologies. *Computer Networks Journal*, 44(6):737–755.

Chang, H., Jamin, S., and Willinger, W. (2003). What causal forces shape Internet connectivity at the AS-level? Technical Report CSE-475-03, EECS Department, University of Michigan.

CIMVP. S. Agarwal, L. Subramanian, J. Rexford and R.H. Katz. Characterizing the Internet hierarchy from Multiple Vantage Points. Project webpage: http://www.cs.berkeley.edu/∼sagarwal/research/BGP-hierarchy/.

Di Battista, G., Patrignani, M., and Pizzonia, M. (2003). Computing the types of the relationships between autonomous systems. In *Proceedings of INFOCOM '03*.

Erlebach, T., Hall, A., Panconesi, A., and Vukadinović, D. (2005). Cuts and disjoint paths in the valley-free path model of Internet BGP routing. In *Proceedings of the First Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN 2004)*. LNCS 3405, pages 49-62, Springer-Verlag.

Erlebach, T., Hall, A., and Schank, T. (2002). Classifying customer-provider relationships in the Internet. In *Proceedings of the IASTED International Conference on Communications and Computer Networks*.

Feigenbaum, J., Papadimitiou, C., Sami, R., and Shenker, S. (2002). A BGP-based mechanism for lowest-cost routing. In *Proceedings of the 21st symposium on principles of distributed computing*. ACM Press, New York.

Gao, L. (2001). On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745.

Gao, L. and Wang, F. (2002). The extent of AS path inflation by routing policies. In *Proceedings of IEEE Global Internet Symposium 2002*.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric algorithms and combinatorial optimization*. Springer-Berlin.

Harary, F. (1969). *Graph Theory*. Addison-Wesley, Massachussets.

Huston, G. (1999a). Interconnection, peering and settlements - Part I. *Internet Protocol Journal*, 2(1):2–16.

Huston, G. (1999b). Interconnection, peering and settlements - Part II. *Internet Protocol Journal*, 2(2):2–23.

Labovitz, C., Ahuja, A., Wattenhofer, R., and Venkatachary, S. (2001). The impact of Internet policy and topology on delayed routing convergence. In *Proceedings of INFOCOM '01*.

LRIP. D.R. Figueiredo, Z. Ge and S. Jaiswal. Logical Relationship Inference Program (implementation of algorithms from Gao, 2001). Webpage: http://www-net.cs.umass.edu/∼ratton/AS/.

NEMECIS. University of California Riverside website. http://ira.cs.ucr.edu:8080/Nemecis.

OREGON. Oregon website: http://www.routeviews.org.

Rimondini, M., Pizzonia, M., Di Battista, G., and Patrignani, M. (2004). Algorithms for the inference of the commercial relationships between autonomous systems: Results analysis and model validation. In *Proceedings of IPS 2004, International Workshop on Inter-domain Performance and Simulation*.

Spring, N., Mahajan, R., and Anderson, T. (2003). Quantifying the causes of path inflation. In *SIGCOMM '03*.

Subramanian, L., Agarwal, S., Rexford, J., and Katz, R. (2002). Characterising the internet hierarchy from multiple vantage points. In *Proceedings of INFOCOM '02*.

Tangmunarunkit, H., Govidan, R., and Shenker, S. (2001a). Internet path inflation due to policy routing. In *Proceedings of SPIE ITcom '01*.

Tangmunarunkit, H., Govidan, R., and Shenker, S. (2003). Internet topology: discovery and policy impact. In Park, K. and Willinger, W., editors, *The Internet as a Large-Scale Complex System*. Oxford University Press.

Tangmunarunkit, H., Govidan, R., Shenker, S., and Estrin, D. (2001b). The impact of routing policy on Internet paths. In *Proceedings of INFOCOM '01*.

Teixeira, R., Marzullo, K., Savage, S., and Voelker, G. (2003). Characterizing and measuring path diversity of Internet topologies. In *Proceedings of SIGMETRICS '03*.

Vanderbeck, F. and Wolsey, L. (1996). An exact algorithm for IP column generation. *Operations Research Letters*, 19:151–159.

Vukadinović, D., Huang, P., and Erlebach, T. (2002). On the spectrum and structure of Internet topology graphs. In *Innovative Internet Computing Systems*, number 2346 in Lecture Notes in Computer Science, pages 83–95.

Xia, J. and Gao, L. (2004). On the evaluation of as relationship inferences. In *Proceedings of IEEE Global Communications Conference (GLOBECOM 2004)*.